DIFFERENCES BETWEEN CONTINUOUS INTEGRATION (CI), DELIVERY (CD), AND DEPLOYMENT



Continuous Integration (CI), Continuous Delivery (CD) and continuous deployment are three distinct but interdependent DevOps practices that define how software moves from development to production. CI focuses on merging and validating code changes frequently; CD extends CI with automated release readiness; and continuous deployment automates the push to production entirely. Together, these practices reduce waste, eliminate "integration hell," and accelerate delivery to end-users.

(This article is part of our DevOps Guide. Use the right-hand menu to navigate.)

Why does traditional software development create problems?

In a conventional development workflow, developers work independently on separate feature branches for weeks or months before merging them together. This delayed integration consistently produces new bugs and conflicts that were never tested for—forcing QA teams to resolve issues under pressure and project managers to absorb unplanned delays.

The result is unhappy developers scrambling to integrate incomplete branches, testers firefighting previously unseen defects, and release schedules that slip. In a previous BMC post, we described the purpose of CI/CD in detail. The CI/CD practices covered here were designed to solve these problems by making integration, testing, and deployment continuous rather than periodic.

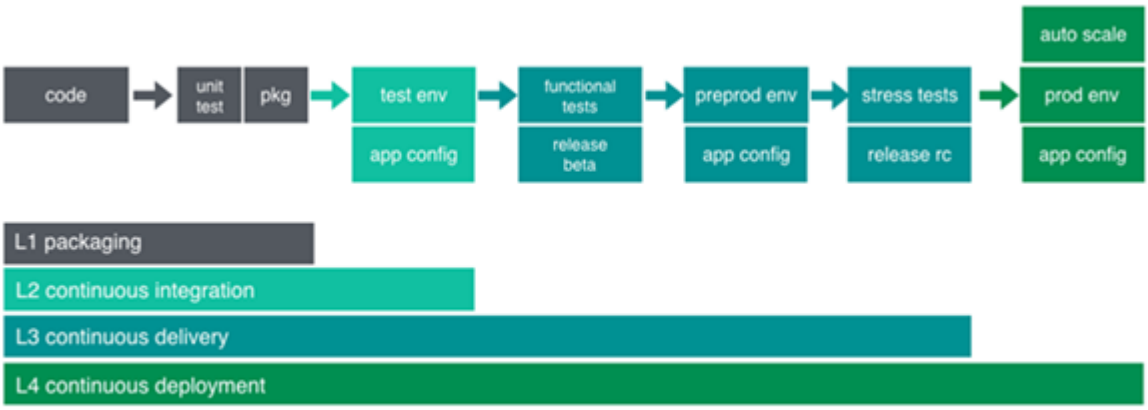


Figure: Continuous deployment maturity model

Source: Testcollab

What is continuous integration (CI)?

Continuous integration is the practice of frequently merging code changes into a centralized shared repository and validating each merge through automated tests. The goal is not to complete a feature branch—it is to ensure all integrated changes work together as a packaged build at all times, with issues surfaced and fixed immediately.

Key attributes of continuous integration:

- Developers write code, run unit tests, and commit packages to a shared repository.
- Code changes are merged frequently—potentially multiple times per day.
- The focus at the CI stage is build stability, not feature completion.
- Automation tools maintain a consistent build and test environment, eliminating the "it worked on my machine" problem and facilitating collaboration across feature branches.
- Feedback on build quality is surfaced early, giving developers immediate context on integration issues before they compound.
- Testing occurs at both the feature branch and mainline branch level.

What is continuous delivery (CD)?

Continuous delivery extends CI by adding automated release capabilities. Each code commit that passes continuous integration is further tested for performance and functionality in a staging environment that replicates production—making it ready for release at any time following a manual approval step.

The distinguishing characteristic of continuous delivery is that the release decision remains a human one—typically a business decision rather than a technical one—but the release process itself is automated, repeatable, and executable with a single action.

Key attributes of continuous delivery:

- Strong continuous integration is a prerequisite for effective continuous delivery.
- The release pipeline is short, automated, and repeatable.
- Automated testing and builds are emphasized throughout.
- Each feature candidate is deployable immediately once approved.
- Releasing a feature to end-users is a business decision, not a technical constraint.

What is continuous deployment?

Continuous deployment is the next evolution of continuous delivery, where a validated build is pushed to production automatically—no manual approval required. Only a failed automated test prevents a change from going live.

Releases in a continuous deployment model are typically targeted at a small subset of end-users initially, generating immediate feedback that flows back to developers for future iterations. This model requires a shared culture and mindset across developers, testers, and operations teams, who must collectively own the full SDLC pipeline and ensure that every small code commit is deployable at a rapid pace.

Key attributes of continuous deployment:

- Deployment to production is fully automated—no human gate.
- All automated tests must pass; a single failure halts the release.
- The mainline branch passes through to the production stage continuously.
- Releases are well-documented at the pace of the release process itself.
- Automation testing is adopted with maximum code coverage.
- Deployment are made in small, frequent batches of feature updates.
- End-users receive continuous changes, often within minutes of a code commit.
- The shortened feedback loop between DevOps teams and end-users reduces operational overhead.

Continuous deployment dramatically compresses the time between writing a code change and end-users experiencing it—from days or weeks to minutes, provided all automated tests pass.

How should DevOps teams choose the right approach?

While continuous integration, continuous delivery, and continuous deployment follow defined principles, each organization adapts these practices to its own team structure, tooling, and business requirements. Not every team needs full continuous deployment—the right release cadence depends on risk tolerance, regulatory requirements, and product maturity.

Every DevOps team must evaluate its specific requirements and design a release strategy grounded in the core differences between these three interdependent CI/CD practices.

Frequently asked questions

What is the main difference between continuous delivery and continuous deployment?

In continuous delivery, a build is release-ready but a human must approve the push to production. In continuous deployment, that approval step is eliminated—any build that passes all automated tests is deployed to production automatically, with no manual intervention.

Does continuous deployment require continuous integration?

Yes. Continuous deployment depends on a strong foundation of continuous integration. Without reliable, automated CI testing, there is no trustworthy signal that a build is safe to deploy automatically to production.

What stops a release in a continuous deployment pipeline?

A failed automated test is the primary gate in a continuous deployment pipeline. If all tests pass, the build is deployed to production without human involvement. Only test failures halt the process.

What does "integration hell" mean in a CI/CD context?

"Integration hell" refers to the painful process of merging long-lived, independent feature branches that have diverged significantly from each other. Continuous integration solves this by requiring developers to merge into a shared repository frequently—often multiple times per day—so conflicts surface early and stay manageable.

Can a team adopt CI without also implementing CD?

Yes. Many teams adopt continuous integration first to improve build quality and catch integration issues early, then layer in continuous delivery or continuous deployment as their automation maturity grows. CI is a prerequisite for CD, but CD is not required to benefit from CI practices.

The views and opinions expressed in this post are those of the author and do not necessarily reflect the official position of BMC.